Approximability of Cycle Covers and Smoothed Analysis of Binary Search Trees

Bodo Manthey

Universität des Saarlandes
manthey@cs.uni-sb.de

Abstract: Der Beitrag enthält eine Zusammenfassung der Dissertation „Approximability of Cycle Covers and Smoothed Analysis of Binary Search Trees“.

Eine Zyklenüberdeckung eines Graphen ist ein Teilgraph, der nur aus Zyklen besteht, so dass jeder Knoten Teil genau eines Zyklus ist. Bei einer L-Zyklenüberdeckung muss zusätzlich die Länge jedes Zyklus in der Menge L liegen. Im ersten Teil der Dissertation wurde die Komplexität und Approximierbarkeit des Problems untersucht, L-Zyklenüberdeckungen maximalen Gewichts zu berechnen. Es wurden einerseits effiziente Approximationsalgorithmen zur Berechnung von L-Zyklenüberdeckungen entwickelt. Andererseits wurde bewiesen, dass L-Zyklenüberdeckungsprobleme für fast alle Mengen L nicht beliebig gut approximiert werden können.

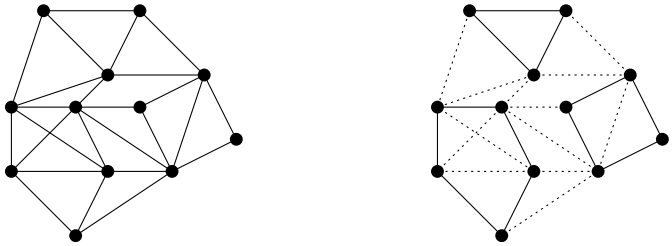
Im zweiten Teil der Dissertation wurde eine Smoothed Analysis der Höhe binärer Suchbäume durchgeführt. Die Smoothed Analysis interpoliert zwischen der Worst-Case-Komplexität, die oft zu pessimistisch ist und durch „pathologische“ Instanzen dominiert wird, und der Average-Case-Komplexität, die oft zu optimistisch ist.

1 Optimierungsprobleme und Approximierbarkeit

Eine Spedition soll Waren zu verschiedenen Kunden ausliefern und möchte dies natürlich mit möglichst geringen Fahrtkosten machen. Ein Tourist möchte alle Sehenswürdigkeiten einer Stadt in möglichst kurzer Zeit besuchen. Hinter diesen und ähnlichen Problemen verbergen sich Optimierungsprobleme. Bei beiden Beispielen handelt es sich um das *Travelling Salesman Problem* (*TSP*), wohl eines bekanntesten Optimierungsprobleme: Eine Instanz des TSP ist ein vollständiger, kantengewichteter Graph, und das Ziel ist es, eine Rundreise durch den Graphen zu finden, die jeden Knoten genau einmal besucht (*Hamiltonscher Zyklus*).

Leider ist das TSP NP-schwer; unter der Annahme NP $\neq$ P gibt es also keinen effizienten Algorithmus, der kürzeste Rundreisen berechnet. In der Praxis ist es oft aber gar nicht notwendig, die kürzeste Rundreise zu berechnen. In vielen Fällen genügt eine Rundreise, die nur wenig schlechter ist als die optimale. Solche Rundreisen schnell (also in polynomieller Zeit) zu berechnen, ist die Aufgabe von *Approximationsalgorithmen*.

Eine Verallgemeinerung Hamiltonscher Zyklen sind *Zyklenüberdeckungen* (*cycle covers*): Eine Zyklenüberdeckung eines Graphen ist eine Menge an Zyklen, so dass jeder Knoten



(a) Ein ungerichteter Graph.

(b) Eine Zyklenüberdeckung des Graphen (durchgezogene Kanten).

Abbildung 1: Beispiel einer Zyklenüberdeckung.

auf genau einem Zyklus liegt. Abbildung 1 zeigt ein Beispiel einer Zyklenüberdeckung. Im Gegensatz zu Hamiltonschen Kreisen können Zyklenüberdeckungen maximalen (oder minimalen) Gewichts effizient berechnet werden. Daher werden Zyklenüberdeckungen oft in Approximationsalgorithmen für das TSP verwendet; tatsächlich basieren die derzeit besten Approximationsalgorithmen für eine ganze Reihe von Varianten des TSP und für verwandte Probleme wie dem *Shortest Common Superstring Problem* auf der Berechnung von Zyklenüberdeckungen.

Im Allgemeinen gilt dabei: Je länger die Zyklen in der berechneten Zyklenüberdeckung sind, desto näher ist die berechnete Lösung am Optimum. Daher möchte man Zyklenüberdeckungen ohne kurze Zyklen berechnen. Es gibt aber auch Approximationsalgorithmen für das TSP, die besonders gute Lösungen liefern, falls die berechneten Zyklenüberdeckungen nur aus Zyklen gerader Länge bestehen. Schließlich basieren Algorithmen zum sogenannten *Vehicle Routing Problem* auf der Berechnung von Zyklenüberdeckungen, die keine zu langen Zyklen enthalten.

Aus diesen Gründen habe ich im ersten Teil meiner Dissertation *eingeschränkte Zyklenüberdeckungen* untersucht, in denen Zyklen bestimmter Längen von vornherein ausgeschlossen sind: Eine L-Zyklenüberdeckung ($L \subseteq \mathbb{N}$) ist eine Zyklenüberdeckung, in der die Länge jedes Zyklus in der Menge L liegt.

Um die Möglichkeiten für Approximationsalgorithmen auszuloten, die auf der Berechnung von Zyklenüberdeckungen basieren, war es das Ziel, diejenigen Mengen L zu charakterisieren, für die L-Zyklenüberdeckungen maximalen (oder minimalen) Gewichts effizient berechnet oder zumindest sehr gut approximiert werden können.

2 Approximierbarkeit von Zyklenüberdeckungen

Ist ein Optimierungsproblem NP-schwer, dann können optimale Lösungen für Instanzen dieses Problems nicht effizient berechnet werden, es sei denn, es wäre $\mathrm{P} = \mathrm{NP}$. Es ist aber noch nichts darüber gesagt, wie gut optimale Lösungen approximiert werden können.

Ein *c-Approximationsalgorithmus* für ein Optimierungsproblem ist ein Polynomialzeit-

algorithmus, der auf Eingabe einer Instanz des Problems stets eine Lösung berechnet, deren Kosten höchstens um einen konstanten Faktor $c \geq 1$ von den Kosten einer optimalen Lösung abweichen. Die Klasse *APX* enthält alle Optimierungsprobleme, für die es einen c-Approximationsalgorithmus gibt. Ist ein Optimierungsproblem APX-schwer, dann gibt es eine Konstante $\varepsilon > 0$, so dass dieses Problem nicht mit Faktor $1+\varepsilon$ approximiert werden kann, es sei denn, es wäre P = NP. In diesem Fall ist also das Berechnen approximativer Lösungen einer bestimmten Güte genauso schwer wie das Finden optimaler Lösungen.

Sei $L \subseteq \mathscr{U} = \{3,4,5,\ldots\}$. (In ungerichteten Graphen haben die kürzesten Zyklen die Länge 3.) Dann ist *L-UCC* das Problem zu entscheiden, ob ein ungerichteter Graph eine L-Zyklenüberdeckung besitzt. *Max-L-UCC* ist folgendes Optimierungsproblem: Die Eingabe ist ein ungerichteter, vollständiger Graph mit Kantengewichten 0 und 1. Das Ziel ist es, eine L-Zyklenüberdeckung maximalen Gewichts zu finden. *Max-W-L-UCC* ist definiert wie Max-L-UCC, der einzige Unterschied ist, dass beliebige natürliche Zahlen als Kantengewichte erlaubt sind.

Sei nun $L \subseteq \mathscr{D} = \{2,3,4,\ldots\}$. (In gerichteten Graphen sind auch Zyklen der Länge 2 möglich.) Dann sind *L-DCC*, *Max-L-DCC* und *Max-W-L-DCC* genauso definiert wie L-UCC, Max-L-UCC bzw. Max-W-L-UCC, abgesehen davon, dass die Instanzen nun gerichtete Graphen sind.

Wir definieren $\overline{L} = \mathscr{U} \setminus L$ im Fall ungerichteter Graphen und $\overline{L} = \mathscr{D} \setminus L$ für gerichtete Graphen (aus dem Kontext wird klar werden, welcher Fall gerade betrachtet wird). Die Menge $\overline{L}$ enthält somit die verbotenen Zyklenlängen.

Ohne Einschränkung der erlaubten Zyklenlängen können alle Varianten des L-Zyklenüberdeckungsproblems in polynomieller Zeit mit Hilfe von Matching-Algorithmen gelöst werden. Auch das Entscheidungsproblem $\overline{\{3\}}$-UCC (nur Länge 3 ist verboten) ist in P.

Hell et al. [2] zeigten, dass L-UCC, und damit auch Max-L-UCC und Max-W-L-UCC, NP-schwer ist, falls $\overline{L} \not\subseteq \{3,4\}$ ist, d.h. falls es eine verbotene Zyklenlänge $\ell \geq 5$ gibt. Max-W-L-UCC ist bereits NP-schwer, wenn es eine verbotene Zyklenlänge $\ell \geq 4$ gibt [2, 7].

Bislang war allerdings kaum etwas über die Komplexität bekannt, L-Zyklenüberdeckungen in gerichteten Graphen zu finden. Des Weiteren war es, abgesehen von einigen Arbeiten zu speziellen Mengen L, unbekannt, wie gut L-Zyklenüberdeckungen approximiert werden können.

2.1 Algorithmen

In meiner Dissertation habe ich einen 2,5-Approximationsalgorithmus für Max-W-L-UCC und einen 3-Approximationsalgorithmus für Max-W-L-DCC entwickelt. Die Laufzeit ist in beiden Fällen $O(n^3)$, wobei n die Anzahl der Knoten ist. Das besondere an diesen Algorithmen ist, dass sie jeweils für alle möglichen Mengen L funktionieren, obwohl es überabzählbar viele davon gibt.

Um dies zu ermöglichen, habe ich zum einen gezeigt, dass es zum Finden approximativer Lösungen genügt, L'-Zyklenüberdeckungen zu berechnen, wobei L' eine (geeignet gewähl-

te) endliche Teilmenge von L ist. Zum anderen habe ich bewiesen, dass jede Zyklenüberdeckung, und damit auch jede L-Zyklenüberdeckung, in knotendisjunkte Pfade (im Fall ungerichteter Graphen) bzw. Kanten (im Fall gerichteter Graphen) zerlegt werden kann, ohne dass man zu viel Gewicht der Zyklenüberdeckung verliert. Die Approximationsalgorithmen lassen sich damit wie folgt kurz zusammenfassen:

1. Berechne eine Zyklenüberdeckung $\tilde{C}$ maximalen Gewichts.
2. Zerlege $\tilde{C}$ in knotendisjunkte Pfade/Kanten P.
3. Füge P zu einer L'-Zyklenüberdeckung C zusammen und gib C aus.

Schließlich habe ich bewiesen, dass Max-$\overline{\{3\}}$-UCC (jeder Zyklus muss also mindestens die Länge 4 haben) in polynomieller Zeit gelöst werden kann.

2.2 Härteresultate

Ich habe gezeigt, dass Max-L-UCC APX-schwer ist für alle L mit $\overline{L} \not\subseteq \{3,4\}$. Darüber hinaus ist Max-W-L-UCC bereits APX-schwer für alle Mengen L mit $\overline{L} \not\subseteq \{3\}$. Dies gilt bereits dann, wenn nur 0, 1 und 2 als Kantengewichte zugelassen sind.

Für gerichtete Graphen konnte ich die Komplexität vollständig klären und folgende Dichotomie zeigen: Für $L = \{2\}$ und $L = \mathscr{D}$ sind L-DCC, Max-L-DCC und Max-W-L-DCC in polynomieller Zeit lösbar, während in allen anderen Fällen L-DCC NP-schwer ist und Max-L-DCC und Max-W-L-DCC APX-schwer sind.

Als Nebenprodukt habe ich bewiesen, dass Min-Vertex-Cover(r) APX-vollständig ist für alle $r \geq 3$. Min-Vertex-Cover(r) ist das Problem, minimale Knotenüberdeckungen in Graphen zu berechnen, in denen aus jedem Knoten r Kanten herausführen. Eine Knotenüberdeckung eines Graphen ist eine Teilmenge der Knoten, so dass jede Kante mindestens einen Endpunkt in dieser Teilmenge hat.

Auf die Beweisidee zu den Härteresultaten wird im nächsten Abschnitt näher eingegangen.

2.3 Clamps und Reduktionen

Wie zeigt man die APX-Härte von überabzählbar vielen Problemen? Natürlich kann nicht für jedes L einzeln eine Reduktion angegeben werden. In der Dissertation habe ich stattdessen das technische Konzept der Rahmenreduktion von Min-Vertex-Cover(r) entwickelt.

Um aus der Rahmenreduktion eine „richtige“ Reduktion zu bekommen, müssen bestimmte Subgraphen, die abhängig von L sind, konstruiert werden. Diese Subgraphen werden dann in die vorgegebene Rahmenkonstruktion eingesetzt. Im Folgenden wird exemplarisch erläutert, welche Eigenschaften die Subgraphen erfüllen müssen, um eine Reduktion auf Max-L-UCC zu erhalten.

Bei diesen Subgraphen handelt es sich um sogenannte *Clamps*: Ein Graph G mit zwei

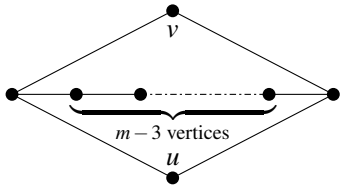


Abbildung 2: Ein L-Clamp für endliche Mengen L mit $\max(L) = m$.

ausgezeichneten Knoten u, v heißt L-Clamp, falls er folgende Eigenschaften erfüllt:

- G_{-v} und G_{-u} besitzen eine L-Zyklenüberdeckung. (G_{-v} entsteht aus G durch Löschen des Knotens v sowie aller zu v inzidenten Kanten.)
- G, G_{-u-v} sowie G^k für alle $k \in \mathbb{N}$ besitzen keine L-Zyklenüberdeckung. (G^k entsteht aus G, indem u und v durch einen Pfad der Länge k verbunden werden.)

Abbildung 2 zeigt ein Beispiel eines L-Clamps. L-Clamps sorgen dafür, dass L-Zyklenüberdeckungen nicht beliebig verlaufen dürfen: Alle Zyklen verlaufen entweder vollständig innerhalb oder vollständig außerhalb des L-Clamps. Die einzige Ausnahme sind die beiden Knoten u und v: Genau einer der beiden Knoten liegt auf einem Zyklus außerhalb des Clamps. Dies entspricht einem exklusiven Oder von u und v und ist die wesentliche Eigenschaft der Clamps, die in der Reduktion benötigt wird.

Die L-Clamps müssen nun auf vollständige Graphen mit Kantengewichten 0 und 1 übertragen werden. Dies geschieht wie folgt: Ist zwischen zwei Knoten eine Kante vorhanden, so weisen wir dieser Gewicht 1 zu. Ist keine Kante vorhanden, dann wird eine Kante mit Gewicht 0 eingefügt.

Was verbleibt, ist die Existenz von L-Clamps nachzuweisen. Hell et al. [2] zeigten, dass es L-Clamps genau dann gibt, wenn $\overline{L} \not\subseteq \{3,4\}$ ist. Zusammen mit der Rahmenreduktion ergibt sich damit die APX-Härte von Max-L-UCC in den genannten Fällen. Die Nichtexistenz von L-Clamps für $\overline{L} = \{4\}$ und $\overline{L} = \{3,4\}$ kann als Hinweis darauf gewertet werden, dass Max-L-UCC in diesen beiden Fällen in polynomieller Zeit lösbar ist.

Ich habe den Begriff L-Clamps auch auf gerichtete Graphen übertragen und konnte zeigen, dass gerichtete L-Clamps für alle L außer $L = \mathscr{D}$ existieren. Für den Beweis der APX-Härte von Max-L-DCC und der NP-Härte von L-DCC wird zusätzlich ein $\ell \in L$ mit $\ell \geq 3$ benötigt. Daraus folgt die APX-Härte von Max-L-DCC für alle L außer $L = \{2\}$ und $L = \mathscr{D}$. Da Max-$\{2\}$-DCC und Max-$\mathscr{D}$-DCC in polynomieller Zeit lösbar sind, ist die Komplexität gerichteter Zyklenüberdeckungen vollständig geklärt.

2.4 Offene Probleme

Tabelle 1 gibt einen Überblick über die Komplexität, eingeschränkte Zyklenüberdeckungen zu berechnen. Es bleibt lediglich die Komplexität von drei Optimierungsproblemen

	L-UCC	Max-L-UCC	Max-W-L-UCC
$\overline{L} = \emptyset$	in P	in PO	in PO
$\overline{L} = \{3\}$	in P	in PO*	
$\overline{L} = \{4\}$			APX-vollständig*
$\overline{L} = \{3,4\}$			APX-vollständig*
sonst	NP-schwer	APX-schwer*	APX-schwer*

(a) Ungerichtete Zyklenüberdeckungen.

	L-DCC	Max-L-DCC	Max-W-L-DCC
$L = \{2\}, L = \mathscr{D}$	in P	in PO	in PO
sonst	NP-schwer*	APX-schwer*	APX-schwer*

(b) Gerichtete Zyklenüberdeckungen.

Tabelle 1: Die Komplexität von Zyklenüberdeckungen. Ergebnisse der Dissertation sind mit „$\star$“ markiert. Die Klasse PO ist die „Optimierungsversion“ der Klasse P.

ungelöst: Max-$\overline{\{4\}}$-UCC und Max-$\overline{\{3,4\}}$-UCC für den Fall, dass nur Gewichte 0 und 1 zugelassen sind, und Max-W-$\overline{\{3\}}$-UCC im Fall beliebiger Kantengewichte. Des Weiteren ist auch die Komplexität der beiden Entscheidungsprobleme $\overline{\{4\}}$-UCC und $\overline{\{3,4\}}$-UCC offen. Eine einfache Lösung der offenen Probleme halte ich für unwahrscheinlich. Es wird vermutet, dass zumindest Max-$\overline{\{4\}}$-UCC und Max-$\overline{\{3,4\}}$-UCC, und damit auch $\overline{\{4\}}$-UCC und $\overline{\{3,4\}}$-UCC, in polynomieller Zeit lösbar sind. Die Komplexität dieser fünf Probleme ist allerdings seit über 20 Jahren offen.

3 Smoothed Analysis

Im ersten Teil der Arbeit wurde die *Worst-Case-Komplexität* der Berechnung von Zyklenüberdeckungen untersucht. Worst-Case-Komplexität bedeutet, dass die Komplexität eines Problems anhand seiner schwierigsten Instanzen gemessen wird. Die Worst-Case-Komplexität hat zwei wesentliche Vorteile: Sie ist oft recht einfach zu bestimmen, und falls sie niedrig ist, dann ist das untersuchte Problem einfach oder der untersuchte Algorithmus arbeitet gut, unabhängig davon, welche Instanzen nun tatsächlich auftreten.

Ein Nachteil der Worst-Case-Komplexität ist, dass sie oft zu pessimistisch ist: Worst-Case-Instanzen sind oft speziell konstruiert, so dass der untersuchte Algorithmus sie möglichst schlecht lösen kann, sie treten aber in der Anwendung gar nicht auf.

Ein weniger pessimistisches Maß ist die *Average-Case-Komplexität*, bei der die Instanzen zufällig gewählt werden und die erwartete Komplexität gemessen wird. Tatsächlich arbeiten viele Algorithmen wesentlich besser auf zufälligen Instanzen. Die Ergebnisse sind aber oft wenig aussagekräftig: Zufällige Instanzen, die die Average-Case-Analyse dominieren, weisen oftmals nicht die Besonderheiten typischer Instanzen auf.

Aber wie fasst man nun mathematisch das Konzept einer „typischen“ Instanz? Um dieses Problem zu lösen und damit auch die Relevanz „pathologischer“ Instanzen, wie sie

beim Untersuchen der Worst-Case-Komplexität auftreten, zu untersuchen, führten Spielman und Teng die sogenannte *Smoothed Analysis* ein [6] (Spielman [5] gibt einen Überblick). Smoothed Analysis interpoliert zwischen Average- und Worst-Case-Komplexität: Es wird die Komplexität von (Worst-Case-)Instanzen gemessen, die leichten, zufälligen Störungen unterworfen werden.

Bei kontinuierlichen Problemen erscheinen gaußverteilte Störungen natürlich. Bei diskreten Problem ist allerdings oft nicht einmal klar, wie „Nähe“ zu definieren ist. Daher müssen Störungsmodelle für diskrete Probleme besonders sorgfältig gewählt werden.

Die Stärke der Störung bei der Smoothed Analysis wird durch einen Parameter p bestimmt. Ist $p = 0$, so wird überhaupt nicht gestört; die Smoothed-Komplexität wird zur Worst-Case-Komplexität. Ist p sehr groß, so verdrängt die Störung die ursprüngliche Instanz, und die Smoothed-Komplexität wird zur Average-Case-Komplexität.

4 Binäre Suchbäume

Binäre Suchbäume sind eine der wichtigsten Datenstrukturen und ein Baustein für viele andere Datenstrukturen. Das Hauptmerkmal für die „Qualität“ eines binären Suchbaums ist die Höhe, d. h. die Länge des längsten Pfades von der Wurzel zu einem Blatt. Ein Binärbaum wird als effizient bezeichnet, falls seine Höhe logarithmisch oder höchstens polylogarithmisch in der Anzahl der Elemente ist. Ist die Höhe linear, dann ist der Baum ineffizient: Der Vorteil von Binärbäumen gegenüber einfachen Listen geht verloren.

Leider ist im Worst-Case die Höhe gleich der Anzahl der Elemente. Andererseits ist der Erwartungswert der Höhe eines zufällig generierten Binärbaums logarithmisch in der Anzahl seiner Elemente. Diese Lücke zwischen Worst- und Average-Case-Höhe zu schließen, war das Ziel des zweiten Teils meiner Dissertation.

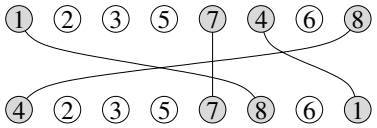
Aus einer Sequenz $\sigma = (\sigma_1, \ldots, \sigma_n)$ von Zahlen erhält man einen *binären Suchbaum* $T(\sigma)$, indem die Elemente iterativ in einen Baum eingefügt werden:

- Die Wurzel von $T(\sigma)$ ist das erste Element σ_1 von σ.
- Seien σ_L die Einschränkung von σ auf Elemente kleiner als σ_1 und σ_R die Einschränkung von σ auf Elemente größer als σ_1. Dann ist $T(\sigma_L)$ der linke und $T(\sigma_R)$ der rechte Teilbaum der Wurzel.

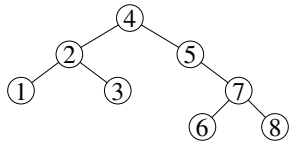
Die Höhe $\text{height}(\sigma)$ des von σ erzeugten Binärbaums $T(\sigma)$ ist die Anzahl der Knoten auf dem längsten Pfad von der Wurzel zu einem Blatt.

Im Worst-Case, z.B. für $\sigma = (1, 2, \ldots, n)$, ist die Höhe eines binären Suchbaums n. Die Average-Case-Höhe unter der uniformen Verteilung ist eines der am besten untersuchten Probleme der Average-Case-Analyse. An dieser Stelle sei dazu nur erwähnt, dass die Average-Case-Höhe $\alpha \ln n + \beta \ln \ln n + O(1)$ ist, wobei $\alpha \approx 4.311$ eine Lösung der Gleichung $\alpha \ln(2e/\alpha) = 1$ und $\beta = \frac{3}{2\ln(\alpha/2)} \approx 1.953$ ist [4].

Um untere Schranken für die Höhe von binären Suchbäumen zu zeigen, habe ich die An-



(a) Oben: $\sigma = (1,2,3,5,7,4,6,8)$. Die grauen Elemente sind markiert. Unten: Das Resultat $\sigma' = (4,2,3,5,7,8,6,1)$.



(b) Der resultierende Baum $T(\sigma')$ der Höhe 4.

Abbildung 3: Eine partielle Permutation.

zahl der Left-To-Right-Maxima einer Sequenz untersucht. Die Anzahl $\mathrm{ltrm}(\sigma)$ der Left-To-Right-Maxima einer Sequenz σ ist die Anzahl der neuen Maxima, die man sieht, wenn man die Sequenz von links nach rechts durchläuft. Es gilt $\mathrm{ltrm}(\sigma) \leq \mathrm{height}(\sigma)$, da die Anzahl der Left-To-Right-Maxima gleich der Länge des rechtesten Pfades in $T(\sigma)$ ist.

Der Worst-Case ist auch hier die sortierte Sequenz: $\mathrm{ltrm}(1,2,\ldots,n) = n$. Unter der uniformen Verteilung erwarten wir $\sum_{i=1}^{n} 1/i \approx \ln n$ Left-To-Right-Maxima. Banderier et al. [1] untersuchten die Anzahl der Left-To-Right-Maxima unter p-partiellen Permutationen (siehe Abschnitt 5.1). Sie bewiesen eine obere Schranke von $O(\sqrt{(n/p)\log n})$ für $0 < p < 1$ und eine untere Schranke von $\Omega(\sqrt{n/p})$ für $0 < p \leq 1/2$.

5 Smoothed Analysis von binären Suchbäumen

Ich habe das Verhalten von Binärbäumen zwischen Worst- und Average-Case untersucht: (Worst-Case-)Sequenzen werden leichten Störungen unterworfen, und es wird die Höhe des Binärbaums gemessen, der durch die gestörte Sequenz erzeugt wird.

5.1 Störungsmodelle

Bevor die eigentliche Smoothed Analysis durchgeführt werden kann, müssen zunächst geeignete Störungsmodelle entworfen werden. Zwei Störungsmodelle erscheinen natürlich: Entweder werden die Positionen einiger Elemente verändert oder die Elemente selbst.

Banderier et al. [1] haben *partielle Permutationen* (*partial permutations*) als Störungsmodell eingeführt. Partielle Permutationen verändern die Position einiger Elemente: Jedes Element wird unabhängig von den anderen mit Wahrscheinlichkeit $p \in [0,1]$ markiert. Anschließend werden alle markierten Elemente zufällig permutiert. Abbildung 3 zeigt ein Beispiel einer partiellen Permutation.

In der Dissertation wurden zwei weitere Störungsmodelle eingeführt: *Partielles Ersetzen* (*partial alterations*) und *partielles Löschen* (*partial deletions*). Wieder wird jedes Element mit Wahrscheinlichkeit p markiert. Beim partiellen Ersetzen wird anschließend jedes markierte Element durch ein Zufallselement ersetzt; sie implementieren also der zweite der

genannten Möglichkeiten, da hier einige Element verändert werden.

Beim partiellen Löschen werden alle markierten Elemente gelöscht. Der Hauptgrund für die Untersuchung von partiellem Löschen ist, dass sich dieses Störungsmodell einfach untersuchen lässt und dass sich die Stabilitätsergebnisse für partielles Löschen auf die anderen beiden Modelle übertragen lassen.

5.2 Obere und untere Schranken

Ich habe asymptotisch scharfe Schranken für die Baumhöhe und die Anzahl der Left-To-Right-Maxima unter allen drei Störungsmodellen gezeigt.

Für jedes $p \in (0,1)$ und alle Sequenzen σ der Länge n ist die erwartete Höhe, wenn auf σ eine p-partielle Permutation angewendet wird, höchstens $6{,}7 \cdot (1-p) \cdot \sqrt{n/p}$. Eine passende untere Schranke erhält man durch Analyse der sortierten Sequenz: Die erwartete Baumhöhe, wenn eine p-partielle Permutation auf die sortierte Sequenz anwendet wird, ist mindestens $0{,}8 \cdot (1-p) \cdot \sqrt{n/p}$. Die untere Schranke ist also asymptotisch gleich der oberen und unterscheidet sich von dieser nur um einen kleinen konstanten Faktor.

Außerdem konnte ich zeigen, dass die Baumhöhe nach einer p-partiellen Permutation mit hoher Wahrscheinlichkeit höchstens $3{,}7 \cdot (1-p) \cdot \sqrt{(n/p) \cdot \log n}$ ist.

Für die Anzahl der Left-To-Right-Maxima konnte ich eine obere Schranke von $3{,}6 \cdot (1-p) \cdot \sqrt{n/p}$ und eine unter Schranke von $0{,}4 \cdot (1-p) \cdot \sqrt{n/p}$ zeigen. Dies verbessert die Ergebnisse von Banderier et al. [1] in drei Aspekten: Die Schranken sind asymptotisch gleich, die Konstante vor der unteren Schranke ist erheblich größer und die Schranken gelten für alle $p \in (0,1)$.

Alle diese Ergebnisse gelten ebenso für p-partielles Ersetzen.

Die Ergebnisse zeigen, dass Worst-Case-Instanzen „zerbrechlich“ sind, wenn man sie stört: Von linearer Worst-Case-Höhe gelangen wir durch leichte Störungen der Sequenzen zu einer erwarteten Höhe von höchstens $O(\sqrt{n})$.

5.3 (In-)Stabilität

Bei der Smoothed Analysis wird analysiert, wie „zerbrechlich“ Worst-Case-Instanzen sind. Ich habe auch eine duale Eigenschaft untersucht: Wie viel schlechter wird die Komplexität von guten Instanz unter Störungen? Anders gefragt: Wie stabil sind Best-Case-Instanzen?

Es zeigt sich, dass es in allen drei Störungsmodellen instabile Best-Case-Instanzen gibt: Es gibt Sequenzen, die Bäume logarithmischer Höhe erzeugen, aber leichtes Stören der Sequenzen liefert Bäume polynomieller Höhe, d.h. Bäume der Höhe $\Omega(n^\delta)$. Natürlich kann im Fall von partiellen Permutationen und partiellem Ersetzen δ nicht größer als $1/2$ sein, da dies der oberen Schranke widersprechen würde. Es kann aber für große p beliebig nahe bei $1/2$ liegen.

6 Ausblick

Bislang wurde Smoothed Analysis fast ausschließlich zur Untersuchung einzelner Algorithmen oder Probleme verwendet. Um allgemeinere Untersuchungen und Klassifikationen zu ermöglichen, benötigt man jedoch eine „Smoothed-Komplexitätstheorie“. Eine besondere Herausforderung scheint dabei die Definition geeigneter Reduktionen zu sein, da diese die Eigenschaften des zugrundeliegenden Störungsmodells berücksichtigen müssen. Bis zu einer allgemeinen Theorie der Smoothed-Komplexität, die auch Konzepte wie Approximierbarkeit berücksichtigt, ist es daher vermutlich noch ein weiter Weg.

Literatur

[1] Cyril Banderier, René Beier und Kurt Mehlhorn. Smoothed Analysis of Three Combinatorial Problems. In: Branislav Rovan und Peter Vojtás (Hrsg.): *Proc. of the 28th Int. Symp. on Mathematical Foundations of Computer Science (MFCS)*, Band 2747 der Reihe *Lecture Notes in Comput. Sci.*, Seiten 198–207. Springer, 2003.

[2] Pavol Hell, David G. Kirkpatrick, Jan Kratochvíl und Igor Kríz. On Restricted Two-Factors. *SIAM J. Discrete Math.*, 1(4):472–484, 1988.

[3] Bodo Manthey. *Approximability of Cycle Covers and Smoothed Analysis of Binary Search Trees*. Dissertation, Universität zu Lübeck, Institut für Theoretische Informatik, 2005.

[4] Bruce Reed. The Height of a Random Binary Search Tree. *J. ACM*, 50(3):306–332, 2003.

[5] Daniel A. Spielman. The Smoothed Analysis of Algorithms. In: Maciej Liśkiewicz und Rüdiger Reischuk (Hrsg.): *Proc. of the 15th Int. Symp. on Fundamentals of Computation Theory (FCT)*, Band 3623 der Reihe *Lecture Notes in Comput. Sci.*, Seiten 17–18. Springer, 2005.

[6] Daniel A. Spielman und Shang-Hua Teng. Smoothed Analysis of Algorithms: Why the Simplex Algorithm Usually Takes Polynomial Time. *J. ACM*, 51(3):385–463, 2004.

[7] Oliver Vornberger. Easy and Hard Cycle Covers. Technischer Bericht, Universität/Gesamthochschule Paderborn, 1980.

Bodo Manthey, geboren am 12. Oktober 1976, legte 1996 am Gymnasium Adolfinum in Bückeburg das Abitur ab. Von 1996 bis 2000 studierte er Informatik mit Nebenfach Bioinformatik/Biomathematik an der Universität zu Lübeck. Danach war er Mitarbeiter am Institut für Theoretische Informatik der Universität zu Lübeck und wurde 2005 promoviert.

Seit März 2006 ist er Assistent an der Universität des Saarlandes in Saarbrücken. Ab August 2006 wird er, unterstützt durch ein Stipendium des DAAD, einen einjährigen Forschungsaufenthalt an der Yale University verbringen.

Im Februar 2006 erhielt Bodo Manthey für seine Dissertation den Staatlichen Universitätspreis der Universität zu Lübeck.